Ask-and-Verify: Span Candidate Generation and Verification for Attribute Value Extraction

Yifan Ding[1*], **Yan Liang**[2], **Nasser Zalmout**[2], **Xian Li**[2], **Christan Grant**[3], **Tim Weninger**[1]
University of Notre Dame[1], Amazon.com[2], University of Oklahoma[3],
{yding4, tweninge}@nd.edu, {ynliang, nzalmout, xianlee}@amazon.com, cgrant@ou.edu

Abstract

The product attribute value extraction (AVE) task aims to capture key factual information from product profiles, and is useful for several downstream applications in e-Commerce platforms. Previous contributions usually formulate this task using sequence labeling or reading comprehension architectures. However, sequence labeling models tend to be conservative in their predictions resulting in a high false negative rate. Existing reading comprehension formulations, on the other hand, can over-generate attribute values which hinders precision. In the present work we address these limitations with a new end-to-end pipeline framework called Ask-and-Verify. Given a product and an attribute query, the Ask step detects the top-K span candidates (*i.e.*, possible attribute values) from the product profiles, then the Verify step filters out false positive candidates. We evaluate Ask-and-Verify model on Amazon's product pages and AliExpress public dataset, and present a comparative analysis as well as a detailed ablation study. Despite its simplicity, we show that Ask-and-Verify outperforms recent state-of-the-art models by up to 3.1% F1 absolute improvement points, while also scaling to thousands of attributes.

1 Introduction

The product profiles in e-Commerce platforms are usually comprised of free-form natural language description of the main product features. The product attribute value extraction (AVE) task is used to extract key factual information from textual product descriptions. Properly extracted attribute values can facilitate several downstream applications, such as search (Xiao et al., 2021), recommendation systems (Hwangbo et al., 2018), and task-oriented dialogue systems (Yan et al., 2017). In the various retail categories there are millions of different product types with thousands of unique attributes, so

Figure 1: An example of attribute value extraction task on a dairy product. Corresponding attribute values are extracted for several different attributes including flavor, target age, brand, gluten, and organic information.

AVE should ideally be scalable with respect to number of attributes, providing high coverage for all possible values, while maintaining accurate overall predictions. Fig. 1 shows an example for the AVE task on a dairy product. In this case, AVE aims to extract the corresponding attribute values of multiple product attributes including *flavor*, *target age*, *gluten information*, among others.

AVE is a central organizational task in online shopping systems, significant attention has been paid to the task resulting in a handful of highly-optimized systems (Zalmout et al., 2021; Zheng et al., 2018; Xu et al., 2019; Yan et al., 2021; Lin et al., 2021; Wang et al., 2020). Most of these models use a sequence labeling formulation or a machine reading comprehension (MRC) formulation. Sequence labeling has been well-known in named entity recognition task. However, its application on AVE task tends to generate conservative outputs, resulting in many false negatives. This is mostly caused by an overabundance of negative token labels (*i.e.*, the 'O' in BIOE schema).

Recently, the AVEQA model (Wang et al., 2020) has tackled the AVE problem from the reading comprehension formulation (*i.e.*, question answering

[*] Most of the work was done during an internship at Amazon.

– hence AVEQA) perspective. This formulation tends to be more flexible and scalable than sequential labelling approaches, however, we observed that AVEQA tends to over-generate irrelevant outputs and does not generalize to multiple attribute values.

In the present work, we address these limitations in existing systems with a new end-to-end framework we dub Ask-and-Verify, consisting of a span candidate generation step (Ask) and a span verification step (Verify). The Ask step first identifies relevant span candidates by locating potential boundaries (*i.e.*starting and ending) with two individual multi-label classifiers based on token features. The goal of the Verify step is to eliminate irrelevant span candidates with span features. The overall framework has an attribute-agnostic nature which can capture salient attribute information from the input sequence, and can generalize to thousands of attributes without attribute-specific parameters.

In summary, we present the Ask-and-Verify framework, which disentangles the attribute value extraction task into an end-to-end pipeline of (1) span candidate generation and (2) verification. We design the multi-label classifiers and span candidate collection module to obtain valid high-quality span candidates within the model. The verification module is an attribute-agnostic binary classifier based on span features. Through extensive experiments on two real-world E-commerce datasets, we show that the Ask-and-Verify framework outperforms the current crop of state-of-the-art models, and is able to scale to thousands of attributes.

2 Related Work

2.1 Attribute Value Extraction

The goal of the attribute value extraction task is to extract key factual information about a product from its text description. Recent contributions typically formulate the AVE task as a sequence labeling task (Karamanolakis et al., 2020; Zheng et al., 2018; Xu et al., 2019; Yan et al., 2021). The main idea is to assign token-wise attribute labels with context-aware token features. To extract different attributes, multiple strategies have been presented by previous works. OpenTag (Zheng et al., 2018) utilized separate tag-sets for each attribute, SuOpenTag (Xu et al., 2019) and Adatag (Yan et al., 2021) utilized single tag-set for all the attributes while attribute information is explicitly injected at the encoder or decoder. Recently, AVEQA (Wang et al., 2020) utilizes machine reading comprehension to extract attribute values by treating attributes as questions and text descriptions as the passage.

2.2 Machine Reading Comprehension

Machine reading comprehension (MRC) is a general task within the fields of information retrieval (IR) and natural language processing (NLP), which aims to find correct answers to a question in a given passage (Rajpurkar et al., 2016, 2018; Zhang et al., 2020, 2021). Illustrated via questions at the bottom of Fig. 1, the AVE task can naturally be formulated as an MRC task, which is to extract correct attribute values (answer) within a product text description (passage) for a given attribute query (question). One complication is that the AVE task must handle unanswerable attribute queries (unanswerable questions) and multiple attribute values for an attribute (multiple answers to a single question).

2.3 Candidate Generation and Selection

Candidate generation and selection is widely used in object detection (Ren et al., 2015; Carion et al., 2020) and instance segmentation (He et al., 2017; Wang et al., 2021) in computer vision. The candidate generation step generates candidate bounding boxes which can carry instance information used in the selection step. Recently, NLP researchers have developed span-based models (Joshi et al., 2020; Yamada et al., 2020; Li et al., 2020; Shen et al., 2021) to obtain state-of-the-art performance on span-based or entity-centered tasks like named entity recognition (Huang et al., 2015; Ding et al., 2021), entity linking (Botzer et al., 2022; Ding et al., 2022), and machine reading comprehension (Rajpurkar et al., 2018) among others. One of the key insights of the Ask-and-Verify framework is to show that this kind-of candidate generation and selection formulation used widely in computer vision can also be used to benefit the the AVE task and potentially other span-level NLP tasks.

3 Methodology

Task Definition: Given a product description $X = [x_1, x_2, ..., x_L]$ with L tokens, and an attribute A from a pre-defined attribute set $\mathcal{A}$, the AVE task aims to extract all of the unique attribute values $\mathcal{Y} = \{y_1, y_2, ..., y_M\}$ corresponding to A. Each attribute value y_m is composed of one or more consecutive tokens within X. If no proper attribute values is found in X for A, an empty set should

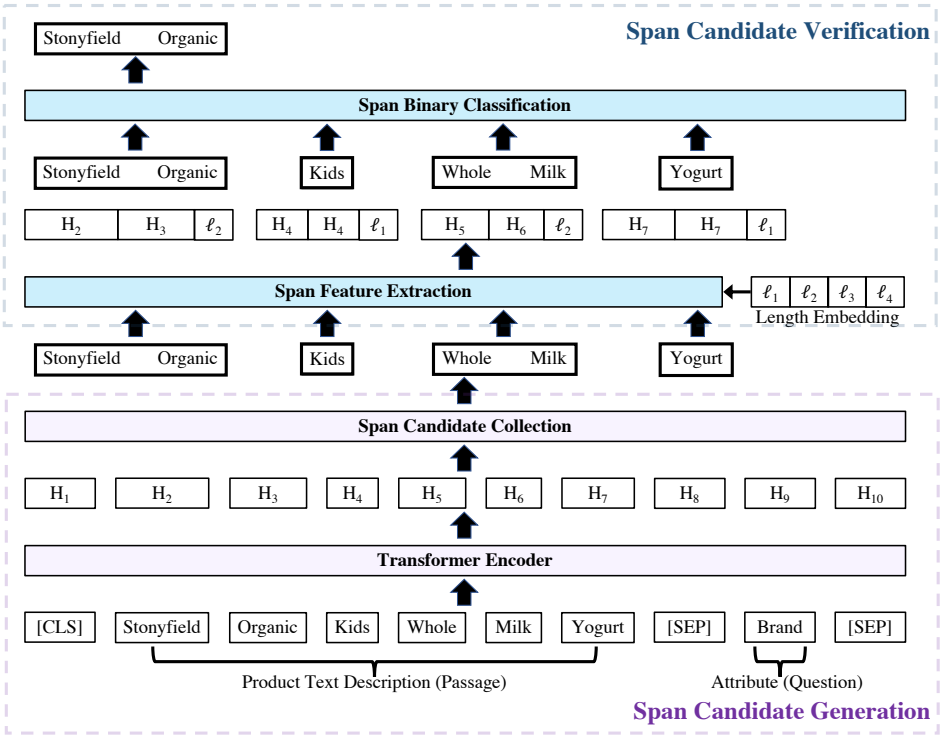

Figure 2: Overview of the Ask-and-Verify framework, a two-step attribute value extraction framework with *Span Candidate Generation* (Ask) and *Span Candidate Verification* (Verify). The framework takes input sequences of entity text descriptions and a single attribute of interest. In the *Span Candidate Generation* step, the input sequence is first passed to a Transformer Encoder to obtain hidden states. The Span Candidate Collection module processes the hidden states to obtain top-K valid span candidates. In the *Span Candidate Verification* step, span embeddings composed of start-token hidden states, end-token hidden states and a span-length embedding, are obtained in Span Feature Extraction module for each generated span candidate. Finally, the span embedding is passed to the Span Binary Classification module to obtain the extracted attribute values.

be returned for $\mathcal{Y}$. Following common practice in question answering, we also call this case as unanswerable case.

Ask-and-Verify: Our framework addresses the AVE task in an end-to-end manner with two major components: (1) span candidate generation (Ask), and (2) span candidate verification (Verify). For an attribute of interest A, the first step generates the potential span candidates. The second verification step filters the candidates and selects a subset.

3.1 Span Candidate Generation

This step generates potential span candidates. Specifically, we employ two individual multi-label classifiers to locate starting index and ending index of span candidates. Formally, given a product with text description X and an attribute of interest A, we use a sub-word tokenizer to tokenize original tokens along with the attribute into sub words SW:

$$SW = \textbf{Tokenizer}(\{\textbf{[CLS]}, X, \textbf{[SEP]}, A\}) \quad (1)$$

Following common practice, we pad the sequences to some fixed length K and longer sequences are also fixed to the same length K by truncating the tokens of text description. The processed tokens are then fed into a **BERT** encoder to obtain d-dimensional hidden states $H \in \mathcal{R}^{K \times d}$:

$$H = \textbf{Encoder}(SW) \quad (2)$$

The hidden states H_s of index s are further fed through a linear layer and Softmax to obtain the probabilities for the starting token. Similarly, the probability index e for the ending token is obtained by feeding the corresponding hidden states H_e through another linear layer and Softmax.

$$P^{\theta}_{\text{start}}(s|X, A) = \frac{\exp(w^T_{\text{start}} H_s)}{\sum_{k=1}^{K} \exp(w^T_{\text{start}} H_k)} \quad (3)$$

$$P^{\theta}_{\text{end}}(e|X, A) = \frac{\exp(w^T_{\text{end}} H_e)}{\sum_{k=1}^{K} \exp(w^T_{\text{end}} H_k)} \quad (4)$$

At the training stage, two K-class (K is the fixed length of input sentence) classifiers are used individually on start indexes and end indexes with multi-label cross-entropy loss (see Eq. (5)-(6)).

Note that start token(s) of $\mathcal{Y}$ forms a set $\mathcal{Y}_S$, and end token(s) of $\mathcal{Y}$ forms a set $\mathcal{Y}_E$. Any correct token in $\mathcal{Y}_S$ is considered as a positive starting token and any correct token in $\mathcal{Y}_E$ is considered as a positive ending token.

$$\mathcal{L}_{\text{start}} = -\sum_{s=1}^{K} \mathbb{1}(SW_s \in \mathcal{Y}_S) \log P^{\theta}_{\text{start}}(s|X, A) \tag{5}$$

$$\mathcal{L}_{\text{end}} = -\sum_{e=1}^{K} \mathbb{1}(SW_e \in \mathcal{Y}_E) \log P^{\theta}_{\text{end}}(e|X, A) \tag{6}$$

To obtain the actual span candidates $\mathcal{M}$, each span candidate $m_{(s,e)}$ has to have top-K probabilities within valid spans (see Eq. (7)). A span is a valid if and only if all the span tokens are within the range of text description with positive lengths up to T tokens (see Eq. (8)). Note that span candidate generation is part of the model thus span candidates are obtained in both training and inference stage.

$$\begin{aligned}\mathcal{M} = \{m_{(s,e)} \mid &\textbf{Mask}_{(s,e)} \wedge \\ &P^{\theta}_{\text{start}}(s|X, A) + P^{\theta}_{\text{end}}(e|X, A) \in \textbf{top-K}\}\end{aligned} \tag{7}$$

$$\begin{aligned}\textbf{Mask}_{(s,e)} = \mathbb{1}(&SW_s \in X \wedge SW_e \in X \\ &\wedge 1 <= e - s <= T)\end{aligned} \tag{8}$$

3.2 Span Candidate Verification

The span candidate verification step aims to verify each span candidate generated from previous step, and choose the final attribute value extraction output. We utilize a simple but effective uniform binary classification model for the verification step. We make individual binary (*i.e.*, yes/no) classifications for each span candidate with the corresponding span-level features.

Formally, given the same hidden state H from Eq. (2), each span candidate $m_{(s,e)}$ obtains its span features $H_{m_{(s,e)}}$ by concatenating starting token's hidden state H_s, ending token's hidden state H_e, and span candidate's length embedding ℓ_{e-s}.

$$H_{m_{(s,e)}} = \text{Concatenate}([H_s; H_e; \ell_{e-s}]) \tag{9}$$

$$\hat{H}_{m_{(s,e)}} = \textbf{DropOut}(\textbf{ReLU}(w_1^T H_{m_{(s,e)}})) \tag{10}$$

$$P^{\theta}_{span}(m_{(s,e)}|X, A) = \frac{\exp(w_2^T \hat{H}_{m_{(s,e)}})}{\sum_{m' \in \mathcal{M}} \exp(w_2^T \hat{H}_{m'})} \tag{11}$$

where $\ell \in \mathcal{R}^{T \times d_F}$ is the learned span length features with d_F dimension. The span features are then fed into a single-layer feed forward neural network with DropOut and a ReLU layer to obtain the corresponding span state $\hat{H}_{m_{(s,e)}}$. $\hat{H}_{m_{(s,e)}}$ further goes through another linear layer and Softmax to obtain the probabilities.

The objective function $\mathcal{L}_{span}$ of the verification step is the sum of the binary cross entropy losses for each span candidate $m_{(s,e)}$. A span is positive if and only if it exactly matches one of the ground truth attribute value(s).

$$\begin{aligned}\mathcal{L}_{span} = -\sum_{m \in \mathcal{M}} \Big(&\mathbb{1}(m \in \mathcal{Y}) \log P(m|X, A) + \\ &\mathbb{1}(m \notin \mathcal{Y}) \log(1 - P(m|X, A)\Big)\end{aligned} \tag{12}$$

$$\mathcal{L} = \mathcal{L}_{start} + \mathcal{L}_{end} + \mathcal{L}_{span} \tag{13}$$

3.3 Training and Inference

The training objective function of the Ask-and-Verify framework is the sum of the starting index loss, ending index loss, and the span binary classification loss (see Eq. (13)). During inference, each span candidate $\hat{m}$ is ranked according to its binary classification score $P^{\theta}_{span}(\hat{m}|X, A)$. The span candidate with higher score than some threshold value τ makes it to the ranking step. In the ranking order, a span candidate is selected if it does not have any overlapping token(s) with any of the already selected spans. If no spans make it to the ranking step, then an empty set is returned. Additional details (*i.e.*hyperparameters) can be found in the reproducibility section A of the appendix.

4 Experiments

We conduct extensive experiments using the AliExpress (Xu et al., 2019) and Amazon datasets and compare the Ask-and-Verify framework with ten state-of-the-art methods.

4.1 Datasets

AliExpress: We use the public version of the AliExpress dataset (**AE-110K**). Following previous work (Wang et al., 2020), we randomly partition the product-instances into an 80/20 train/test split for scaling experiments. Additionally, we also focus on the 50 most frequent attributes, but remove 2

Table 1: Test macro precision (P), recall (R) and F1 scores on AE-48, AZ-15 and AZ-33. Best scores are highlighted in bold, second best scores are underlined.

Model	AE-48			AZ-15			AZ-33		
	P	R	F_1	P	R	F_1	P	R	F_1
BiLSTM (Huang et al., 2015)	0.788	0.771	0.776	0.742	0.519	0.593	0.731	0.511	0.586
BERT (Devlin et al., 2019)	0.787	**0.814**	0.800	0.720	0.506	0.582	0.736	0.509	0.589
OpenTag (Zheng et al., 2018)	-	-	-	0.751	0.519	0.594	0.708	0.482	0.557
SuOpenTag (Xu et al., 2019)	0.806	0.795	0.798	0.749	0.503	0.585	0.711	0.533	0.593
AdaTag (Yan et al., 2021)	0.801	0.805	0.799	0.751	0.518	0.591	0.712	0.542	0.599
AVEQA (Wang et al., 2020)	0.806	0.807	0.804	0.618	0.512	0.551	0.633	0.523	0.563
MRC-For-NER (Li et al., 2020)	0.753	0.800	0.774	0.562	0.428	0.470	0.652	0.482	0.543
W2NER (Li et al., 2022)	-	-	-	**0.847**	0.304	0.405	**0.838**	0.271	0.369
Locate-and-Label (Shen et al., 2021)	0.713	0.673	0.669	0.655	**0.564**	0.569	0.697	0.512	0.549
Sequence-to-Set (Tan et al., 2021)	0.778	0.621	0.665	0.786	0.411	0.501	0.763	0.420	0.501
Ask-and-Verify	**0.821**	0.813	**0.814**	0.750	0.551	**0.625**	0.744	**0.562**	**0.629**

that fail on the AdaTag model (Yan et al., 2021). This setting is referred to as **AE-48** dataset.

Amazon: Similar to previous work (Yan et al., 2021), we collected datasets from Amazon's product pages. The raw training data includes 33 frequent attributes and 745,216 total samples. Example attributes including color, flavor, skin type, hair type, pattern type, and age range description. Test data is annotated by Amazon employees, including 15 attributes (from the 33 attributes) and 11,000 total samples. We consider two experiment settings: first we use all 33 attributes (**AZ-33**) from the training set; in the second setting we restrict the training instances to include at least one of 15 attributes present in test set (**AZ-15**).

4.2 Existing Models

We compared the AVE task performance of the Ask-and-Verify against ten state-of-the-art models including two standard sequential labeling models: BiLSTM (Huang et al., 2015) and BERT (Devlin et al., 2019); four state-of-the-art attribute value extraction models: OpenTag (Zheng et al., 2018), SuOpenTag (Xu et al., 2019), AdaTag (Yan et al., 2021), and AVEQA (Wang et al., 2020); four state-of-the-art named entity recognition models: MRC-for-NER (Li et al., 2020), W2NER (Li et al., 2022), Locate-and-Label (Shen et al., 2021), and Sequence-to-Set (Tan et al., 2021).

4.3 Metrics

Following previous work we use the exact entity matching criteria for evaluation. A predicted attribute value is considered to be a true positive if and only if it exactly matches one of the ground truth values. In the main experiments shown in Table 1 we compute macro precision (P), recall (R) and F1 scores (F1) by aggregating across testing attributes. In the scaling experiment shown in Table 2, we use micro precision, recall and F1 scores following previous work (Wang et al., 2020).

Table 2: Test micro precision (P), recall (R) and F1 scores on AE-110k. Many models are not included because they could not scale to the large size of attributes in the AE-110K dataset. Best scores are highlighted in bold. ♣: our reported scores are different from original reported values, details can be found in section B and D of appendix.

Model	AE-110K		
	P	R	F_1
SuOpenTag (Xu et al., 2019)	0.641	0.575	0.607
AVEQA (Wang et al., 2020) ♣	0.784	0.711	0.746
Ask-and-Verify	**0.798**	**0.723**	**0.759**

4.4 Results

The results of our principal experiments over the three E-Commerce dataset settings (AE-48, AZ-15 and AZ-33) are listed in Table 1. We observe that the Ask-and-Verify framework outperforms the existing methods on all three settings with absolute improvements in the F1 score of +1.0%, +3.1% and +3.0% respectively, compared to second best baseline. Specifically, Ask-and-Verify obtains the best or second best recall scores. As for precision, only state-of-the-art NER models obtain higher precision than Ask-and-Verify but with the cost of much

Table 3: Ablation study of different model choices on span candidate generation (Ask) and verification (Verify). Best scores are highlighted in bold.

	AZ-15			AZ-33		
	P	**R**	**F_1**	**P**	**R**	**F_1**
Ask-and-Verify	0.750	0.551	**0.625**	0.744	0.562	**0.629**
w/o Verify	0.279	**0.601**	0.374	0.328	**0.618**	0.421
Verify w/ PURE	**0.825**	0.316	0.418	**0.829**	0.287	0.388
Ask w/ n-gram	0.717	0.510	0.587	0.708	0.507	0.582
Ask w/ nouns	0.611	0.264	0.361	0.580	0.266	0.357

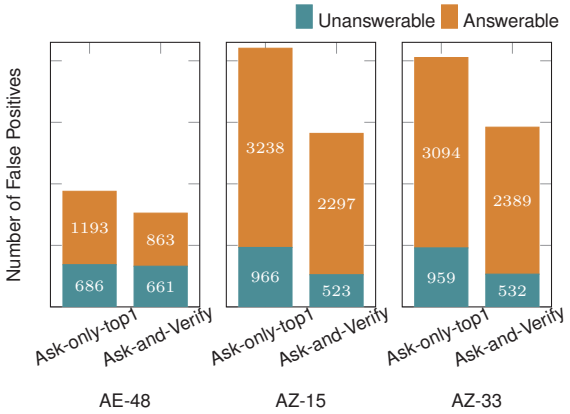

Figure 3: Number of false positive samples in Ask-only and Ask-and-Verify settings. The Verify-step substantially reduces false positives.

more downgrade on the recall.

We further conduct experiments to test the scalability of the Ask-and-Verify framework on the AE-110K setting with more than two thousand attributes. Results of this experiment are listed in Table 2. We compared Ask-and-Verify with AVEQA and SuOpenTag only because the other models are not able to scale up to all attributes. We find the AVEQA model shows significantly better scalability compared to the SuOpenTag. Compared to AVEQA, the Ask-and-Verify framework is able to further boost the performance, with improvements in precision, recall and F1 scores at +1.4%, +1.2% and 1.3% respectively.

5 Ablation Studies

5.1 Effectiveness of the Ask-Step

We study the effect of using alternative formulations for the Ask step to provide span candidates. Intuitively, we can remove the Ask component and replace it with a basic n-grams setup. Specifically, we use n-grams and noun phrase chunking as alternatives for the Ask step. For the n-grams model, we consider all the possible spans up to five words. For the noun-phrase chunking model, we utilize spaCy's chunker[1] to extract all the nouns of the input text. We keep the same Verification step and conduct experiments on AZ-15 and AZ-33 settings. As shown in Table 3, both n-grams and noun phrase chunking show a lower performance compared to the Ask component. Using n-grams drops f1 by 3.8% on AZ-15, and 4.7% on AZ-33. While noun phrases result in a larger drop, by more than 20% on both settings.

5.2 Effectiveness of the Verification-Step

To study the effect of the Verification step, we first consider removing the Verification step completely, and use the span candidates directly as the output. As shown in Table 3, removing the verification step leads to significant precision drop of about 47.1% and 41.6%, with moderate recall improvement of 5.0% and 5.6% on the AZ-15 and AZ-33. We also replace the Verification step with the PURE (Zhong and Chen, 2021) model. Even though the alternative can improve precision with 7.5% and 8.5%, it drops recall by about 23.5% and 27.5%.

Further, to quantitatively understand how the Verify model can reduce the number of false positives presented in the span candidates, we count the false positives in the Ask-and-Verify output and Ask-only-top1 span output. Compared to Ask-only-top-1, the addition of the Verify model in our framework can significantly reduce the false positives in both answerable and unanswerable cases. In the AE-48 setting, answerable false positive samples are reduced by about 20%. On the AZ-15 and AZ-33 settings, Ask-and-Verify can filter-out more than 40% of the unanswerable cases, and more than 20% of the answerable cases.

6 Deployment Considerations

The Ask-and-Verify framework is currently under deployment evaluation. Investigation is going on to verify if the framework can be deployed with minimal changes of the existing workflow. In our deployment tests, we found that the Ask-and-Verify framework has better precision and recall performance. Ask-and-Verify is also flexible, and can adapt to a wide application scenarios that might require varying precision and recall levels, by changing the threshold values. Moreover, a single model can cover a large number of attributes – this is par-

[1] https://github.com/explosion/spaCy

ticularly important since E-commerce platforms can hold billions of different products with thousands of attributes.

7 Conclusions

In this paper we described a new end-to-end framework, Ask-and-Verify, for the attribute value extraction task. This framework has two main components: (1) a span candidate generation step, and (2) a verification step. The span candidate generation step can provide high-quality span candidates and the verification step can further remove irrelevant span candidates. Ask-and-Verify utilizes two individual multi-label classifiers in the candidate span generation step and an attribute agnostic span-based binary classifier in the verification step. We performed a comparative analysis on an Amazon products dataset as well as a publicly available dataset from AliExpress. We evaluate Ask-and-Verify compared to ten other baseline models. Despite its simplicity, Ask-and-Verify consistently outperforms these state-of-the-art methods, and is able to scale up to thousands of unique attributes. Ask-and-Verify also has high flexibility and allows for effective threshold tuning.

Acknowledgements

This project was funded in part by DARPA under contract HR001121C0168 and HR00112290106. We would like to thank Chenwei Zhang and Jun Ma from Amazon, for their constructive feedback and inspiration on the work. We would also like to thank the anonymous reviewers for their valuable comments.

References

Nicholas Botzer, Yifan Ding, and Tim Weninger. 2022. Reddit entity linking dataset. *Information Processing and Management*, 58(3).

Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. 2020. End-to-end object detection with transformers. In *2020 European Conference on Computer Vision*, pages 213–229, Cham. Springer International Publishing.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Ning Ding, Guangwei Xu, Yulin Chen, Xiaobin Wang, Xu Han, Pengjun Xie, Haitao Zheng, and Zhiyuan Liu. 2021. Few-NERD: A few-shot named entity recognition dataset. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3198–3213, Online. Association for Computational Linguistics.

Yifan Ding, Nicholas Botzer, and Tim Weninger. 2022. Posthoc verification and the fallibility of the ground truth. In *Proceedings of the First Workshop on Dynamic Adversarial Data Collection*, pages 23–29, Seattle, WA. Association for Computational Linguistics.

Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. 2017. Mask r-cnn. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988.

Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional lstm-crf models for sequence tagging. *arXiv:1508.01991*.

Hyunwoo Hwangbo, Yang Sok Kim, and Kyung Jin Cha. 2018. Recommendation system development for fashion retail e-commerce. *Electronic Commerce Research and Applications*, 28:94–101.

Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S. Weld, Luke Zettlemoyer, and Omer Levy. 2020. SpanBERT: Improving pre-training by representing and predicting spans. *Transactions of the Association for Computational Linguistics*, 8:64–77.

Giannis Karamanolakis, Jun Ma, and Xin Luna Dong. 2020. TXtract: Taxonomy-aware knowledge extraction for thousands of product categories. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8489–8502, Online. Association for Computational Linguistics.

Jingye Li, Hao Fei, Jiang Liu, Shengqiong Wu, Meishan Zhang, Chong Teng, Donghong Ji, and Fei Li. 2022. Unified named entity recognition as word-word relation classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 10965–10973.

Xiaoya Li, Jingrong Feng, Yuxian Meng, Qinghong Han, Fei Wu, and Jiwei Li. 2020. A unified MRC framework for named entity recognition. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5849–5859, Online. Association for Computational Linguistics.

Rongmei Lin, Xiang He, Jie Feng, Nasser Zalmout, Yan Liang, Li Xiong, and Xin Luna Dong. 2021. Pam: Understanding product images in cross product category attribute extraction. page 3262–3270.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789, Melbourne, Australia. Association for Computational Linguistics.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc.

Yongliang Shen, Xinyin Ma, Zeqi Tan, Shuai Zhang, Wen Wang, and Weiming Lu. 2021. Locate and label: A two-stage identifier for nested named entity recognition. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2782–2794, Online. Association for Computational Linguistics.

Zeqi Tan, Yongliang Shen, Shuai Zhang, Weiming Lu, and Yueting Zhuang. 2021. A sequence-to-set network for nested named entity recognition. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*, pages 3936–3942. International Joint Conferences on Artificial Intelligence Organization. Main Track.

Qifan Wang, Li Yang, Bhargav Kanagal, Sumit Sanghai, D. Sivakumar, Bin Shu, Zac Yu, and Jon Elsas. 2020. Learning to extract attribute value from product via question answering: A multi-task approach. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*, page 47–55, New York, NY, USA. Association for Computing Machinery.

Yuqing Wang, Zhaoliang Xu, Xinlong Wang, Chunhua Shen, Baoshan Cheng, Hao Shen, and Huaxia Xia. 2021. End-to-end video instance segmentation with transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8741–8750.

Liqiang Xiao, Jun Ma, Xin Luna Dong, Pascual Martinez-Gomez, Nasser Zalmout, Wei Chen, Tong Zhao, Hao He, and Yaohui Jin. 2021. End-to-end conversational search for online shopping with utterance transfer. *arXiv preprint arXiv:2109.05460*.

Huimin Xu, Wenting Wang, Xinnian Mao, Xinyu Jiang, and Man Lan. 2019. Scaling up open tagging from tens to thousands: Comprehension empowered attribute value extraction from product title. In *ACL*.

Ikuya Yamada, Akari Asai, Hiroyuki Shindo, Hideaki Takeda, and Yuji Matsumoto. 2020. Luke: deep contextualized entity representations with entity-aware self-attention. *arXiv preprint arXiv:2010.01057*.

Jun Yan, Nasser Zalmout, Yan Liang, Christan Grant, Xiang Ren, and Xin Luna Dong. 2021. Adatag: Multi-attribute value extraction from product profiles with adaptive decoding. In *ACL*.

Zhao Yan, Nan Duan, Peng Chen, Ming Zhou, Jianshe Zhou, and Zhoujun Li. 2017. Building task-oriented dialogue systems for online shopping. In *Thirty-First AAAI Conference on Artificial Intelligence*.

Nasser Zalmout, Chenwei Zhang, Xian Li, Yan Liang, and Xin Luna Dong. 2021. All you need to know to build a product knowledge graph. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 4090–4091.

Zhuosheng Zhang, Junjie Yang, and Hai Zhao. 2021. Retrospective reader for machine reading comprehension. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 14506–14514.

Zhuosheng Zhang, Hai Zhao, and Rui Wang. 2020. Machine reading comprehension: The role of contextualized language models and beyond. *arXiv preprint arXiv:2005.06249*.

Guineng Zheng, Subhabrata Mukherjee, Xin Luna Dong, and Feifei Li. 2018. Opentag: Open attribute value extraction from product profiles. In *KDD*.

Zexuan Zhong and Danqi Chen. 2021. A frustratingly easy approach for entity and relation extraction. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 50–61.

A Reproducibility

Ask-and-Verify is implemented using the PyTorch and transformers packages based on the uncased BERT model. We use the transformers' trainer and default AdamW optimizer with the learning rate setting to $3e^{-5}$ in all the experiments. Training epoch and batch size vary for different datasets (see Table 4). The experiments on AZ-15 and AE-33 takes about 18 hours to run on 4 NVIDIA TITAN XP GPUs. The experiments on AE-48 and AE-110E take less than six hours on single GPU.

	dataset	AE-48	AE-110E	AZ-15	AZ-33
Ask	batch-size	48	48	48	96
	epoch	30	30	5	30
Verify	batch-size	128	128	128	128
	epoch	100	30	10	10

Table 4: batch size and epoch used by Ask-and-Verify on different experiments

The Ask-and-Verify framework has a handful of hyper-parameters. The mentioned values are applicable for all the experiments unless specifically indicated. The maximum number of sub-words for a span candidate is set to 5. The verification model considers at most 5 span candidates in both training and inference stages. Both *span candidate generation* and *verification* input a list of tokens consisting of a text description, a single attribute of interest, and special tokens. The length (*i.e.*number of tokens) of the input sequence K is set to 512. Shorter sentences are padded with a special token. Longer sentences are truncated. In the verification step, the dimension of the span length features d_F is set to 150 and the dropout rate is set to 0.2. The inference threshold τ uses the default value of binary classifier 0.5.

B Preprocessing and Postprocessing

Attribute value extraction is formalized as different tasks by different methods in our experiments, including machine reading comprehension (*i.e.*, question answering), sequence labeling, and *span candidate generation and verification*. There can be multiple ways to conduct preprocessing and postprocessing in these tasks. For example, the labels of MRC can be defined on word-level (before tokenization) or sub-word-level (after tokenization). We tried to ask for the preprocessing code used for the AVEQA (Wang et al., 2020) paper, but were unable to obtain it. Following previous sequence labeling work (Zheng et al., 2018; Yan et al., 2021), we first prepare tokens and associated labels in the sequence labeling format for each experiment setting. Then labels of other formats are transformed from the sequence labeling format in the preprocessing step. Output results are later transformed back to sequence labeling format to conduct evaluation. Specifically, we required all the possible extracted attribute values being a subset of continuous full words within the input sequence X for each method. In comparison, standard sub-word tokenizer (*e.g.*BERT) can generate sub-words not necessarily forming full words. For example, if a word is called "swimglass" and sub-word tokenizer can generate "swim" only. We observe the differences have impacts on the evaluation results of different experiment settings.

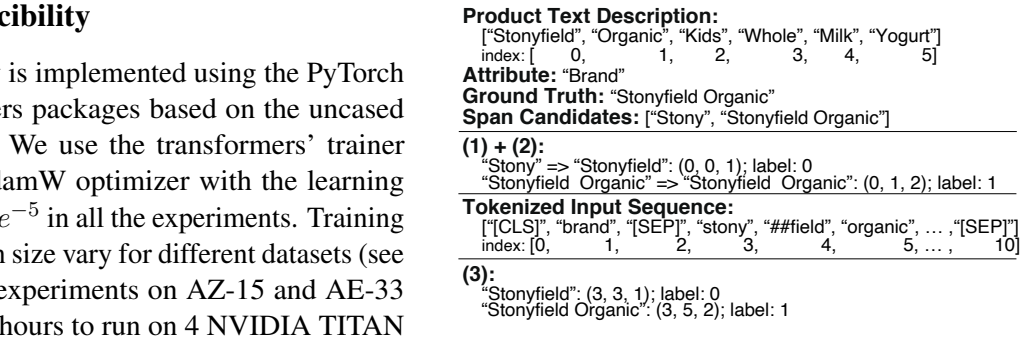

Figure 4: An example of processing span candidates.

C Span Candidate Process

In Ask-and-Verify, a span candidate $m_{(s,e)}$ is represented as a tuple (with start-index s, end-index e, span-length $\ell = e - s$). The span candidate must consist of continuous context sub-words with no more than the predefined maximum number of sub-word tokens. Additionally, the span candidates from the generation step are composed of sub-words which do not necessarily form full words. This setting may cause extra errors in final evaluation. Furthermore, the index s and e are not the same as original span candidate's positions because of sub-word tokenization and the attribute injected in the input sequence. To overcome the interface challenge, we present a processing pipeline in the span candidate collection module to: (1) locate or transform span candidates to nearest tokens forming in the full-word formats; (2) assign binary classification labels with strict string matching between the processed span candidates and ground

Table 5: Models that address the attribute value extraction task and their features on M product attribute types.

	# of models	Scales to AE-110K	Negative Words
OpenTag	M		✓
SuOpenTag	1	✓	✓
AdaTag	2		✓
AVEQA	1	✓	
Ask-and-Verify	1	✓	✓

truth (only in training phase); and (3) capture the correct position of sub-word tokens corresponding to the start token and end token. An example is illustrated in Fig. 4.

D AE-110K dataset

We utilize the public version of AE-110K dataset. However, the data split process, pre-processing, post-processing and evaluation code are not public released. We observed our experiment values for AVEQA (74.6% F1) and SuOpenTag (60.7% F1) are both lower than the values reported in the AVEQA paper: AVEQA (85.01% F1) and SuOpenTag (74.92% F1). This is most likely coming from different data splits and pre-processing strategies.

E Contribution Matrix

In the attribute value extraction task, we argue that there are three major dimensions to judge a framework in the industry production environment: efficiency, scaling ability, and performance. A good framework should have few number of models, capable of scaling up to large number of attributes while obtaining good performances as shown in the Table 5. Compared to all the previous methods, Ask-and-Verify has only one model for multiple attributes, scaling up to thousands of attributes on the public AE-110K dataset, and also carefully considering negative words resulting superior performances on two real word datasets.

F Precision-Recall curve

We present the precision-recall curve of the Ask-and-Verify model on the AZ-15 and AZ-33 settings by changing the threshold values of the Verify step. The performances of other baseline methods are also included in the same figures. From the results in Fig. 5, we can see that the precision and recall keeps a high performance score in a wide range. Compared to the performance of other baselines, the precision and recall curve of Ask-and-Verify

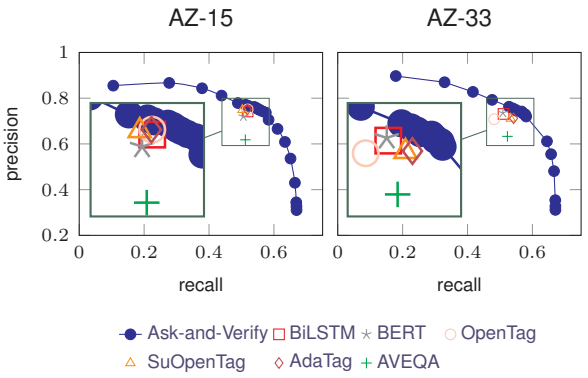

Figure 5: precision and recall curves of Ask-and-Verify on AZ-15 and AZ-33 settings. Ask-and-Verify has a good performance and high flexibility. Both precision-recall curves are above all the comparing methods.

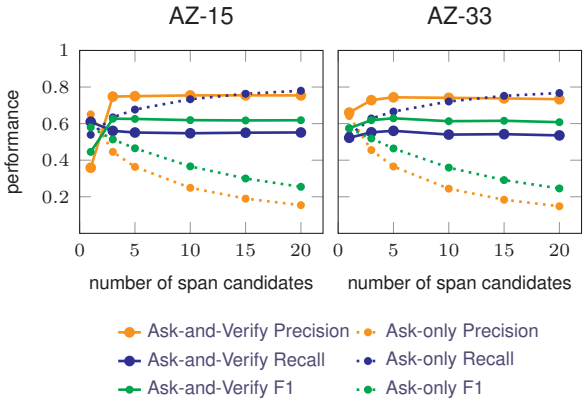

Figure 6: Ablation study on changing the number of span candidates on AZ-15 and AZ-33 settings. With number of span candidates increase, the ask-only provides spans with higher recall but lower precision. In comparison, Ask-and-Verify model has a moderate increase from 1 to 5 span candidates and keeps a stable performance with 5 or more span candidates.

is always on the top right. Interestingly, the margin gets larger on the AZ-33 compared to AZ-15, showing the better scalability of Ask-and-Verify.

G Effectiveness of Changing the Number of Span Candidates

Span candidate plays a central role in the framework by bridging Ask model and Verify model. Intuitively, increasing the number of span candidates can potentially include extra positive span samples but also bring more negative samples at the same time. It is also interesting to investigate how the trade offs impact the performances of overall architecture. Fig. 6 shows the metrics of Ask-only and

Ask-and-Verify models by generating or utilizing the same number of span candidates. We first analyze the Ask-only performances from the dash lines. When the number of span candidates increases, we can see that the recall curve is constantly increasing while precision and f1 curves keep decreasing. Considering the top-20 span candidates, the recall can be even larger than 75% while the precision is only around 25%. As for the Ask-and-Verify performance represented with solid lines, it shows interesting patterns. With top-1 span candidates on AZ-15 setting, it has slightly larger precision and slightly lower recall compared to Ask-only outputs. As the number of span candidates increases to 5, both precision and recall increase. As the number of span candidates keeps increasing, both precision and recall drop slightly but still keep stable scores.

H Case Study

We present some examples to show how Ask-and-Verify span candidate verification step can better capture the correct attribute values from the span candidates. These examples are illustrated in Fig. 7 (A-D). The AVE task of (A) is to extract the "skin tone" attribute from the tanning product. The Ask model produces span candidates with "darker", "softening and tan extending DHA", "all" and "black", sorted by the ranking scores of the Ask model. The "darker" span candidate ranks first and is the output of the Ask-only model. However, the verification model chooses the third-ranked span candidate "all" as the output, matching the ground truth. The AVE task of example (B) is to extract the "hair type" attribute from a Hair Conditioner product. All span candidates are incorrect and Ask-and-Verify is able to reject all the irrelevant attribute values. Example (C) seeks to extract the same "hair type" attribute from a gel product. Even if the correct attribute values are not included within the span candidates, Ask-and-Verify can still reject each candidate and therefore reduce false positives. Finally, in example (D) we seek to extract the "Age Range" attribute from a pack of diapers. Ask-and-Verify can correctly identify the "Baby" attribute value in the second span candidate.

In summary, we find that the Ask-step predicts frequent and contextually-coherent span candidates. However, these span candidates carry many false positives Introducing the verification-step into the framework appears to substantially reduce the occurrence of false positives.

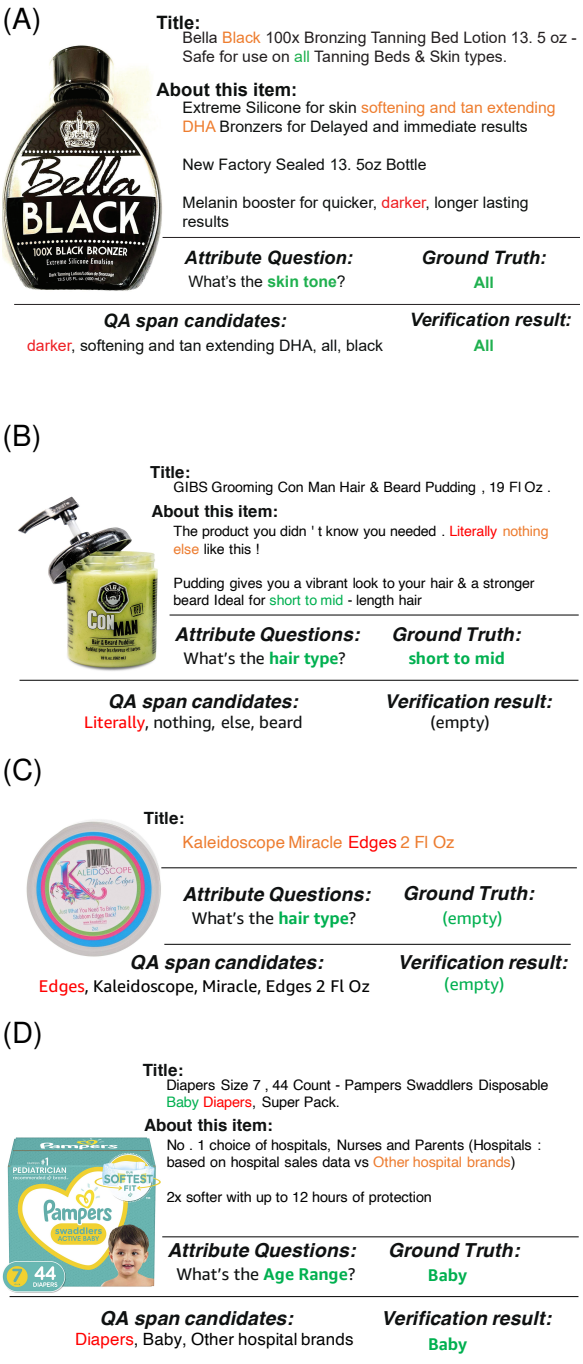

Figure 7: Case study with four illustrative examples of (A) tanning lotion, (B) hair conditioner, (C) hair gel, and (D) diapers. We find that the Ask-step of the Ask-and-Verify model is able to produce reasonable, but noisy candidates each of the product attributes. However, the Verify-step is able to filter-out spurious candidates and reduce the rate of false positives.